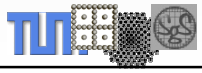
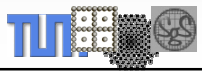
Grid Generation and Refinement

- numerical treatment of PDE requires approximate description and discretization of the resp. domain
- main tasks: *generating* and *refining* grids or meshes; closely related with discretization of PDE:
  - *point* discretization, cf. finite differences
  - *cell* discretization, cf. finite elements or volumes
- objectives:
  - **accuracy**: accurate (and dense) enough to catch the essential physical phenomena
  - **boundary approximation**: sufficiently detailed to represent boundaries and boundary conditions
  - **computational efficiency**: small overhead for handling of data structures, no loss of performance on supercomputers
  - **numerical adequacy**: features with a negative impact on numerical efficiency should be avoided (angles, distortions)



Basic Types of Grids

- we distinuguish *structured* and *unstrutured* grids:
  - **structured**: construction of points or elements follows some regular process from which *geometric* (coordinates) and *topological* information (neighbour relations) can be derived
  - **unstructured**: completely irregular generation, even random choice is possible, which requires explicit storage of basic geometric and topological information
- issues of interest:
  - **grid generation**: initial placement of grid points or elements
  - **grid adaptation**: need for grid points often becomes clear only during the computations, requires possibilities of both **refinement** and **coarsening**
  - **grid partition**: standard parallelization techniques are based on some *subdivision* or *decomposition* of the underlying domain



Structured Grids 1

- rectangular meshes:
  - prototype, bricks or cubes
  - restricted with respect to complexity of domain
- composite grids:
  - subdivide (complicated) domain into subdomains of simpler form and use regular meshs there
  - *block* or *patched* grids: glue subregions together along interfaces (with or without continuity)
  - *overlaid* or *chimera* grids: subdomain grids are completely independent and do not fit together (overlap, interpolation)
- block-structured grids:
  - subdivision into *logically* rectangular subdomains (with logically rectangular local grids)
  - subdomains fit together in an unstructured way, but continuity is ensured (coinciding grid points)

Structured Grids 2

- PDE generators: based on the idea of a transformation from the unit square/cube to the computational domain
  - elliptic: grid coordinates are obtained as solutions of a system of elliptic PDE

$$\Delta\xi(x,y) = 0 \quad \text{on} \quad]0,1[^2$$
$$\Delta\eta(x,y) = 0 \quad \text{on} \quad]0,1[^2$$

    - boundary conditions:
      $(\xi,\eta) =$ shape of the computational domain's boundary
    - ensures very smooth grids, even if boundaries are not smooth
    - explicit grid control (exact position of points) is difficult
    - in nonconvex case, lines may leave domain
  - inverse elliptic:
    - Laplacians are defined on the computational (curvilinear) domain
    - no external lines, but now a more complicated system to be solved

Structured Grids 3

- PDE generators (continued):
  - **hyperbolic**:
    - solve hyperbolic system; for physically unbounded domains
  - **algebraic**:
    - interpolation-based, no extra PDE to be solved
    - most famous representative: **transfinite interpolation** or **Coons patch** (the c-curves denote the domain's boundary)

$$F_1(x,y) = (1-x)\cdot c(0,y) + x\cdot c(1,y)$$
$$F_2(x,y) = (1-y)\cdot c(x,0) + y\cdot c(x,1)$$
$$F_{12}(x,y) = (1-x)(1-y)\cdot c(0,0) + x(1-y)\cdot c(1,0) + (1-x)y\cdot c(0,1) + xy\cdot c(1,1)$$
$$TF(x,y) = (F_1 + F_2 - F_{12})(x,y)$$

    - interpolation of boundary curves into interior, same in 3D
    - cheap, easy to control; non-smooth grids and leaving lines possible

Unstructured Grids 1

- closely related to FEM, typically triangles/tetrahedra
  - **Delaunay triangulations**:
    - suppose you already have grid points – how to define elements?
    - go back to **Dirichlet** and **Voronoi** and define *Voronoi regions* around each given grid point:

$$V_i = \left\{P : \|P - P_i\| < \|P - P_j\| \;\; \forall j \neq i\right\}$$

    - result is called a *Voronoi diagram* (subdivision of domain into polygons or polyhedra, resp.)
    - draw mid-lines between all pairs of neighbouring points; leads to set of disjoint triangles or tetrahedra
    - very widespread
  - **point generation**: how to get grid points?
    - superimpose a regular grid and refine (quadtree, octree)
    - or: start with some boundary point distribution, generate Delaunay triangulation, continue with subdivision following suitable rules
    - if helpful, add point or lines sources (singularities, bound. layers)

Unstructured Grids 2

- advancing front methods:
  - start from the boundary (*starting front*) and advance step by step into the interior:
    - choose an edge on the current front, say PQ
    - create a new point R at equal distance from P and Q
    - determine all grid points lying within a circle around R, radius r
    - order these points w.r.t. distance from R
    - for all these points, form triangles with P and Q, select one
    - if accepted (no intersections etc.), add to list of points
    - add new edges, change front line
- spacetree with boundary fitted closure:
  - create a quadtree/octree of given accuracy, add triangles or tetrahedra, resp., at boundary
- hybrid grids: many mixed forms

Adaptive Refinement

- adaptivity with structured grids:
  - block-wise refinement, interpolation at hanging nodes
  - hierarchical approach (quadtrees, octrees)
- adaptivitiy with unstructured grids: more popular
  - *global error estimator* : tells us whether the result computed so far is sufficiently accurate
  - *refinement criterion* : determines the aim of refinement (balancing of error over grid points, keeping error below some threshold everywhere, ...)
  - *local error estimator or indicator* : tells us during computation where to refine the grid locally
  - *refinement procedure* : determines the technical process of refinement (centroid, red, green)

Varying Geometries

- applications:
  - free surface problems (injection moulding, melting, freezing)
  - multiphase flows
  - fluid-structure interactions
- common strategies:
  - **front tracking methods**:
    - describe boundary or interface explicitly, update of geometry due to movements
    - accurate, but expensive; topology changes?
  - **front capturing methods**:
    - follow interface indirectly (some global quantity)
    - less precise, but more straightforward
    - examples: Volume-of-Fluid, Marker-and-Cell
  - **sliding mesh techniques**:
    - part of the grid moves, the other part is fixed (cf. ALE approach)